Random Thoughts on Not Much of Anything

CH2005/06/13

1. **Software power**

   1.1. Yes, of course, you can do anything in software. And yes, of course, you can simulate anything.
   1.2. No, you cannot defeat physics or statistics. Not even with the best software or the most sophisticated simulation. If you want your work to have anything to do with reality, that is.

2. **Anybody can simulate**

   2.1. It takes a smart guy to know what he is simulating.
   2.2. It takes a wise guy to know what he is not simulating.
   2.3. Invariably, people just do simulation.

3. **On software engineering projects**

   3.1. Software engineering is invariably planned and managed based on a well-defined unit: smop (=small matter of programming). Every task requires exactly one unit of smop, independent of the task size and the stage of the project.
   3.2. The only variable is the reality coefficient that scales a smop to mundane quantities such as time and money. The scaling coefficient for smop is a monotonically and rapidly increasing function of time and project stage.

4. **On software engineering and testing**

   4.1. Software engineering is 5% implementation and 95% testing. And in 95% of the time spent testing, you work on the wrong thing.
   4.2. The longer you spend debugging a problem, the shorter the solution. Vice versa.
   4.3. If there is a 50/50 chance that you are right the first time, then there is a 90% chance that you'll be wrong the first time. In trying to correct a wrong, there is a 95% chance that your first fix is wrong.
   4.4. There are two major types of bugs:
      - Type A: you spend 95% time finding the bug and 5% time fixing it.
      - Type B: you spend 5% time finding the bug and 95% time wasting energy until you realize that there really is no fix.
   4.5. Good news is that 95% of the bugs are type A. Bad news is that type B is only identifiable at the very end. Occasionally, you get lucky. You spend 5% time finding the problem and 5% time solving it. Then you spent 90% the time congratulating yourself and commenting: "It's always good to find a problem when you have a problem."
   4.6. It takes a smart guy to know what he is testing. It takes a wise guy to know what he is not testing. Invariably, people just do testing.

5. On software maintenance

5.1. The number of hours it takes to get an old code running (I mean really running) is equal to the number of weeks since you last run it.
5.2. The only way to maintain a code is to have a person run it all the time. And there is only one person that can do the job: you, the person who wrote the code.

6. How do you tell it’s working?

6.1. It is straightforward to prove a positive and impossible to prove a negative.
6.2. In software development, it is infinitely more preferable to have a problem than not to have one. Just think, which one is harder to prove: 'Software is broken' or 'Software is working?'

7. Ball in your court

7.1. Item 6 also applies to hardware. But the hardware guys can always blame the software.
7.2. In the rare occasions when the hardware guys admit to a problem, they always ask the software guys to fix it in software. The software guys, based the universal belief system listed in 1, will invariably do that just to prove their superiority. Along the way, three more problems will inevitably be created.
7.3. After 7.2, it becomes a software problem.

8. Tip of an iceberg

8.1. There is always an iceberg, even when you don't see the tip. Trust me, it's there.
8.2. Experience says: the smaller the tip, the bigger the iceberg.
8.3. Based on 8.2, the iceberg whose tip you don't see is infinitely big.
8.4. Based on 8.3, no software actually works. Then again, I can't prove it.